Parte NOVE
JDBC e Localização

Capítulo VINTE E NOVE
API JDBC

Objetivos do Exame

- Descrever as interfaces que compõem o núcleo da API JDBC, incluindo as interfaces Driver, Connection, Statement e ResultSet, e sua relação com implementações de provedores.
- Identificar os componentes necessários para conectar-se a um banco de dados usando a classe DriverManager, incluindo a URL JDBC.
- Enviar consultas e ler resultados do banco de dados, incluindo a criação de instruções, retorno de conjuntos de resultados, iteração através dos resultados e fechamento apropriado de conjuntos de resultados, instruções e conexões.

Introdução

A API de Conectividade de Banco de Dados Java (ou JDBC) define como podemos acessar um banco de dados relacional. Você pode encontrar suas classes e interfaces no pacote java.sql.

A versão mais recente desta API, incluída no Java 8, é a JDBC 4.2.

O principal benefício de usar JDBC é que ela oculta os detalhes da implementação entre bancos de dados ao usar um conjunto de interfaces, para que você não precise escrever códigos diferentes para acessar bancos de dados diferentes.

Este capítulo assume que você sabe como usar SQL para selecionar, inserir, atualizar e deletar registros de uma entidade de banco de dados.

Além disso, para este capítulo, é recomendado (embora não exigido) ter acesso a um banco de dados para praticar e testar alguns conceitos. No entanto, ele não ensinará como instalar ou usar um.

Se você não tem um banco de dados instalado (ou acesso a um), talvez a maneira mais fácil seja usar o **JavaDB**, um banco de dados que é fornecido com o Java e que pode ser usado como um banco de dados embutido ou servidor de rede. Você pode encontrá-lo no diretório db da instalação do JDK.

Interfaces JDBC

Ao usar JDBC, você trabalha com interfaces em vez de implementações. Essas implementações vêm de um driver JDBC.

Um driver é apenas um arquivo JAR com classes que sabem como se comunicar com um banco de dados específico. Por exemplo, para MySQL, existe o mysql-connector-java-XXX.jar, onde XXX é a versão do banco de dados.

Mas você não precisa saber como as classes dentro do driver são implementadas ou nomeadas, você só precisa conhecer as quatro principais interfaces que elas precisam implementar:

java.sql.Driver

Todo driver JDBC deve implementar esta interface para saber como se conectar ao banco de dados.

java.sql.Connection

A implementação fornece métodos para obter informações sobre o banco de dados, criar instruções e gerenciar conexões.

java.sql.Statement

A implementação é usada para executar instruções SQL e retornar resultados.

java.sql.ResultSet

A implementação é usada para recuperar e atualizar os resultados de uma consulta.

Além dessas interfaces, uma classe importante é java.sql.DriverManager, que mantém o controle dos drivers JDBC carregados e obtém a conexão real com o banco de dados.

Conectando a um Banco de Dados

Primeiramente, para trabalhar com um banco de dados, você precisa se conectar a ele. Isso é feito usando a classe DriverManager.

Antes de tentar estabelecer uma conexão, a classe DriverManager precisa carregar e registrar quaisquer implementações de java.sql.Driver (que sabem como estabelecer a conexão).

Talvez você já tenha visto em algum programa uma linha como esta:

```java
Class.forName("com.database.Driver")
```

Isso carrega a implementação do Driver para que o DriverManager possa registrar o driver.
No entanto, isso não é mais necessário desde o JDBC 4.0, porque o DriverManager carrega automaticamente qualquer driver JDBC 4.0 presente no classpath.

Uma vez que os drivers estão carregados, você pode se conectar a um banco de dados com o método estático DriverManager.getConnection():

```java
Connection getConnection(String url)
Connection getConnection(String url, Properties info)
Connection getConnection(String url, String user, String passw)
```

Esse método irá procurar entre os drivers registrados para ver se consegue encontrar um que possa fazer uma conexão usando a URL fornecida.

A URL varia dependendo do banco de dados utilizado. No entanto, elas têm três partes em comum:

Formato da URL JDBC

1. **Protocolo** (sempre o mesmo)
2. **Subprotocolo** (na maioria das vezes o nome do banco de dados/tipo do driver)
3. **Propriedades específicas de conexão do banco de dados** (na maioria das vezes a localização do banco de dados com o formato: //SERVIDOR:PORTA/NOME_DO_BANCO_DE_DADOS)

A primeira parte é sempre a mesma: jdbc.
A segunda parte, na maioria das vezes, é o nome do banco de dados e/ou o tipo do driver, como mysql, postgresql, oracle:thin.
A última parte varia de acordo com o banco de dados, mas na maioria das vezes, ela contém o nome (ou IP) do host, a porta e o nome do banco de dados ao qual você está se conectando, como //192.168.0.1:3306/db1.

Aqui estão alguns exemplos de URL:

```java
jdbc:postgresql://localhost/test
jdbc:sqlserver://localhost\SQLEXPRESS;databasename=db
jdbc:derby:db;create=true
```

Podemos nos conectar ao banco de dados chamando este método para obter um objeto Connection:

```java
Connection con =
    DriverManager.getConnection("jdbc:mysql://localhost/db");
```

Se você precisar passar um usuário ou senha para autenticação, podemos usar:

```java
Properties props = new Properties();
props.put("user", "db_user");
props.put("password", "db_p4assw0rd");
Connection con =
    DriverManager.getConnection("jdbc:mysql://localhost/db", props);
```

Ou:

```java
Connection con =
    DriverManager.getConnection("jdbc:mysql://localhost/db",
                                "db_user",
                                "db_p4assw0rd");
```

Antes que seu programa termine, você precisa fechar a conexão (caso contrário, você pode ficar sem conexões).

Felizmente, Connection implementa AutoCloseable, o que significa que você pode usar um **try-with-resources** para fechar automaticamente a conexão:

```java
String url = "jdbc:mysql://localhost/db";
String user = "db_user";
String passw = "db_p4assw0rd";

try(Connection con =
        DriverManager.getConnection(url, user, passw)) {
    // Operações de banco de dados
} catch(SQLException e) {
    System.out.format("%d-%s-%s",
                      e.getErrorCode(),
                      e.getSQLState(),
                      e.getMessage());
}
```

Uma SQLException é lançada sempre que há um erro no JDBC (como quando o driver para se conectar ao banco de dados não é encontrado no classpath), e muitos métodos da API JDBC a lançam, então ela precisa ser capturada (ou recuperada no caso de exceções suprimidas).

Como essa exceção é usada para muitos erros, para saber o que deu errado, você deve usar getMessage(), que retorna uma descrição do erro, getSQLState() que retorna um código de erro padrão, ou getErrorCode(), que retorna o código específico do banco de dados para o erro.

Agora que temos um objeto Connection, podemos executar algumas instruções SQL.

Executando Consultas

Você precisa de um objeto Statement para executar consultas e realizar operações no banco de dados.

Existem três interfaces Statement:

java.sql.Statement

Representa instruções SQL simples para o banco de dados, sem parâmetros.

java.sql.PreparedStatement

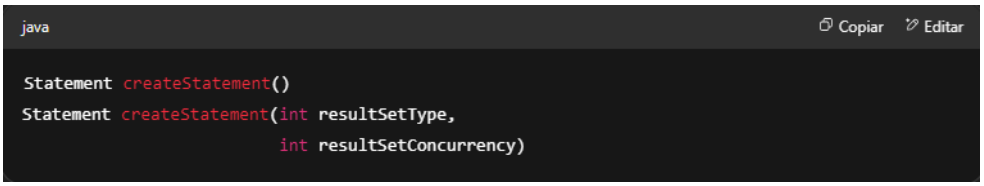
Representa instruções SQL pré-compiladas para executar instruções múltiplas vezes de forma eficiente. Pode aceitar parâmetros de entrada.

java.sql.CallableStatement

Usado para executar procedimentos armazenados. Pode aceitar parâmetros de entrada, bem como de saída.

Na prática, PreparedStatement é o mais utilizado, mas para o exame, você só precisa conhecer Statement.

Você pode obter um Statement a partir de um objeto Connection usando o método createStatement():



Ao criar um Statement, você pode definir o tipo do conjunto de resultados e seu modo de concorrência.

Existem três tipos de conjuntos de resultados:

ResultSet.TYPE_FORWARD_ONLY

Este é o tipo padrão. Quando especificado, você só pode percorrer os resultados uma vez e na ordem em que foram recuperados.

ResultSet.TYPE_SCROLL_INSENSITIVE

Quando especificado, você pode avançar e retroceder pelos resultados e ir a uma posição específica no conjunto de resultados.

ResultSet.TYPE_SCROLL_SENSITIVE

Quando especificado, você também pode avançar, retroceder e ir a uma posição específica no conjunto de resultados, mas sempre verá as alterações mais recentes nos dados ao usá-lo, em contraste com TYPE_SCROLL_INSENSITIVE, que **não é** "sensível" a alterações nos dados.

Na prática, a maioria dos drivers não oferece suporte a TYPE_SCROLL_SENSITIVE. Se você solicitar e ele não estiver disponível, receberá TYPE_FORWARD_ONLY ou (mais provavelmente) TYPE_SCROLL_INSENSITIVE.

Existem dois modos de concorrência:

ResultSet.CONCUR_READ_ONLY

Este é o modo padrão. Quando especificado, você não pode atualizar (usando uma instrução INSERT, UPDATE ou DELETE) um conjunto de resultados.

ResultSet.CONCUR_UPDATABLE

Indica que o conjunto de resultados pode ser atualizado.

Se você solicitar o modo CONCUR_UPDATABLE e seu driver não o suportar, você pode receber o modo CONCUR_READ_ONLY.

Na maioria das vezes, os valores padrão são usados:

```java
Statement stmt = con.createStatement();
```

Agora que temos um objeto Statement, temos três métodos à nossa disposição para executar comandos SQL:

Método	Instruções SQL Suportadas	Tipo de Retorno
execute()	SELECT, INSERT, UPDATE, DELETE, CREATE	boolean (retorna true para SELECT, false para os demais)
executeQuery()	SELECT	ResultSet
executeUpdate()	INSERT, UPDATE, DELETE, CREATE	Número de linhas afetadas (zero para CREATE)

Um objeto Statement deve ser fechado, mas assim como Connection, ele implementa AutoCloseable, então pode ser usado também com try-with-resources:

```java
try(Connection con =
      DriverManager.getConnection(url, user, passw);
    Statement stmt = con.createStatement()) {
   boolean hasResults = stmt.execute("SELECT * FROM user");
   if(hasResults) {
      // Para recuperar o objeto com os resultados
      ResultSet rs = stmt.getResultSet();
   } else {
      // Para obter o número de linhas afetadas
      int affectedRows = stmt.getUpdateCount();
   }
   ResultSet rs = stmt.executeQuery("SELECT * FROM user");
   stmt.executeUpdate("INSERT INTO user(id, name) "
        + "VALUES(1, 'George')"); // Retorna 1
   stmt.executeUpdate("UPDATE user SET name='Joe' "
        + "WHERE id = 1"); // Retorna 1
   stmt.executeUpdate("DELETE FROM user "
        + "WHERE id = 1"); // Retorna 1
} catch(SQLException e) {
   e.printStackTrace();
}
```

Escolha o método execute() correto com base no tipo da instrução SQL que você está usando, porque se você usar o método errado, uma SQLException será lançada.

Quando você usa uma instrução SELECT, pode ler o resultado através de um objeto ResultSet, que veremos a seguir.

Lendo Resultados

Um objeto ResultSet é usado para ler os resultados de uma consulta em formato tabular (linhas contendo as colunas especificadas).

Esse objeto mantém um cursor que aponta para a linha atual, e você só pode ler uma linha por vez.

No início, o cursor está logo antes da primeira linha. Chamar o método next() irá avançar o cursor uma posição e retornar true se houver dados, ou false se não houver. Dessa forma, você pode iterar em um loop por todo o conjunto de resultados.

Assim como Connection e Statement, um objeto ResultSet também precisa ser fechado e também implementa AutoCloseable:

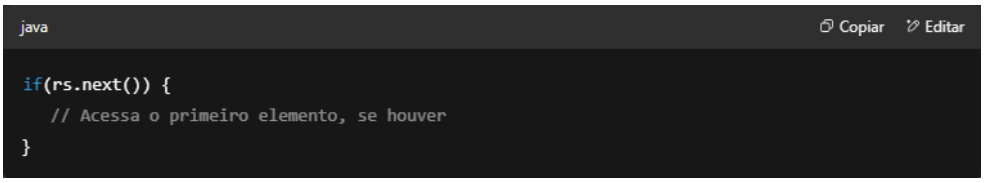
É uma boa prática fechar esses recursos dessa forma, mas não é obrigatório. Aqui estão as regras para fechamento de recursos JDBC:

- O objeto ResultSet é fechado primeiro, depois o objeto Statement, e depois o objeto Connection.
- Um ResultSet é fechado automaticamente quando outro ResultSet é executado a partir do mesmo objeto Statement.
- Fechar um Statement também fecha o ResultSet.
- Fechar um Connection também fecha os objetos Statement e ResultSet.

Se a consulta não retornar resultados, um objeto ResultSet ainda será retornado (embora next() retorne false na primeira chamada).

Lembre-se, next() não apenas informa se há mais registros a serem processados, ele também **avança o cursor** para a próxima linha.

Isso significa que mesmo quando você deseja acessar apenas a primeira linha do resultado, ainda precisa chamar next() (de preferência com uma instrução if, porque se não houver elementos, uma SQLException será lançada se você tentar acessar qualquer coisa):



Agora, para realmente obter os dados, ResultSet possui métodos getter para vários tipos de dados, por exemplo:

- getInt() retorna um int
- getLong() retorna um Long
- getString() retorna uma String
- getObject() retorna um Object
- getDate() retorna um java.sql.Date
- getTime() retorna um java.sql.Time

- getTimestamp() retorna um java.sql.Timestamp

Para cada método, há duas versões:

- Uma que recebe uma String que representa o nome da coluna (isso **NÃO** é sensível a maiúsculas/minúsculas).
- Outra que recebe um int que representa o índice da coluna de acordo com a ordem declarada na cláusula SELECT. A primeira coluna começa em 1, não 0.

Por exemplo:

```java
ResultSet rs = stmt.executeQuery(
      "SELECT id, name FROM user"
);
while(rs.next()) {
   int id = rs.getInt("id");
   String name = rs.getString("name");
   // Faça algo
}
```

É equivalente a:

```java
ResultSet rs = stmt.executeQuery(
      "SELECT id, name FROM user"
);
while(rs.next()) {
   int id = rs.getInt(1);
   String name = rs.getString(2);
   // Faça algo
}
```

Se referenciarmos uma coluna inexistente (seja por índice ou nome), uma SQLException será lançada.

Observe que os métodos getDate(), getTime() e getTimestamp() não retornam objetos de data ou hora padrão, então eles podem precisar ser convertidos, por exemplo:

```java
ResultSet rs = stmt.executeQuery(
      "SELECT insertion_date FROM user"
);

while(rs.next()) {
   // Obtendo a parte da data
   java.sql.Date sqlDate = rs.getDate(1);
   // Obtendo a parte da hora
   java.sql.Time sqlTime = rs.getTime(1);
   // Obtendo ambos, data e hora
   java.sql.Timestamp sqlTimestamp =
                       rs.getTimestamp(1);
   // Convertendo data
   LocalDate localDate = sqlDate.toLocalDate();
   // Convertendo hora
   LocalTime localTime = sqlTime.toLocalTime();
   // Convertendo timestamp
   Instant instant = sqlTimestamp.toInstant();
   LocalDateTime localDateTime =
                sqlTimestamp.toLocalDateTime();
}
```

Portanto, é assim que você trabalha com conjuntos de resultados TYPE_FORWARD_ONLY.

Ao trabalhar com conjuntos de resultados roláveis (TYPE_SCROLL_INSENSITIVE ou TYPE_SCROLL_SENSITIVE), temos muitas opções para mover o cursor.

Aqui está a lista completa de métodos para movimentar o cursor (apenas lembre-se de que todos os métodos, exceto next(), exigem um conjunto de resultados rolável):

Método	**Descrição**
boolean absolute(int row)	Move o cursor para o número de linha dado no conjunto de resultados, contando a partir do início (se o argumento for positivo) ou do fim (se negativo).Se o argumento for zero, o cursor é movido para antes da primeira linha.Retorna true se o cursor for movido para uma posição válida ou false se o cursor estiver antes da primeira linha ou depois da última.
void afterLast()	Move o cursor para depois da última linha.
void beforeFirst()	Move o cursor para antes da primeira linha.
boolean first()	Move o cursor para a primeira linha.Retorna true se o cursor estiver em uma linha válida ou false se não houver linhas no conjunto de resultados.
boolean last()	Move o cursor para a última linha.Retorna true se o cursor estiver em uma linha válida ou false se não houver linhas no conjunto de resultados.
boolean next()	Move o cursor para a próxima linha.Retorna true se a nova linha atual for válida ou false se não houver mais linhas.
boolean previous()	Move o cursor para a linha anterior.Retorna true se a nova linha atual for válida ou false se o cursor estiver antes da primeira linha.
boolean relative(int rows)	Move o cursor um número relativo de linhas, positivo ou negativo.Movimentar além da primeira/última linha posiciona o cursor antes/depois da primeira/última linha.Retorna true se o cursor estiver em uma linha válida, false caso contrário.

Considerando a seguinte tabela:

ID	**NOME**	**DATA_INSERÇÃO**
1	THOMAS	2016 / 03 / 01
2	LAURA	2016 / 03 / 01
3	MAX	2016 / 03 / 01
4	KIM	2016 / 03 / 01

O programa a seguir mostra alguns desses métodos. Tente acompanhá-lo:

```java
try(Connection con =
        DriverManager.getConnection(
                                url, user, passw);
     Statement stmt = con.createStatement(
              ResultSet.TYPE_SCROLL_INSENSITIVE,
              ResultSet.CONCUR_READ_ONLY);
     ResultSet rs = stmt.executeQuery(
              "SELECT * FROM user")) {
   System.out.println(rs.absolute(3)); // true
   System.out.println(rs.getInt(1)); // 3
   System.out.println(rs.absolute(-3)); // true
   System.out.println(rs.getInt(1)); // 2
   System.out.println(rs.absolute(0)); // false
   System.out.println(rs.next()); // true
   System.out.println(rs.getInt(1)); // 1
   System.out.println(rs.previous()); // false
   System.out.println(rs.relative(2)); // true
   System.out.println(rs.getInt(1)); // 2
   System.out.println(rs.relative(0)); // true
   System.out.println(rs.getInt(1)); // 2
   System.out.println(rs.relative(10)); // false
   System.out.println(rs.previous()); // true
   System.out.println(rs.getInt(1)); // 4
} catch(SQLException e) {
   e.printStackTrace();
}
```

Pontos-Chave

- A API de Conectividade de Banco de Dados Java (ou JDBC) define como podemos acessar um banco de dados relacional. Você pode encontrar suas classes e interfaces no pacote java.sql.
- Ao usar JDBC, você trabalha com interfaces em vez de implementações. Essas implementações vêm de um driver JDBC. As quatro principais interfaces a serem implementadas são:
    - java.sql.Driver
    - java.sql.Connection
    - java.sql.Statement
    - java.sql.ResultSet
- Além dessas interfaces, uma classe importante é java.sql.DriverManager, que mantém o controle dos drivers JDBC carregados e obtém a conexão real com o banco de dados.
- Uma vez que os drivers estão carregados, você pode se conectar a um banco de dados com o método estático:

```java
DriverManager.getConnection(String url)
```

- A URL varia dependendo do banco de dados usado. No entanto, elas têm três partes em comum:
    - A primeira parte é sempre a mesma: jdbc.
    - A segunda parte, na maioria das vezes, é o nome do banco de dados e/ou o tipo do driver, como mysql, postgresql, oracle:thin.

    - A última parte varia de acordo com o banco de dados, mas na maioria das vezes, ela contém o nome (ou IP) do host, a porta e o nome do banco de dados ao qual você está se conectando, como: //192.168.0.1:3306/db1.

- Uma vez que temos um objeto Connection, podemos executar algumas instruções SQL obtendo um objeto Statement.
- Você pode obter um Statement a partir de um objeto Connection usando o método createStatement().
- Ao criar um Statement, você pode definir o tipo do conjunto de resultados e seu modo de concorrência.
- Existem três tipos de conjuntos de resultados:
    - ResultSet.TYPE_FORWARD_ONLY
    - ResultSet.TYPE_SCROLL_INSENSITIVE
    - ResultSet.TYPE_SCROLL_SENSITIVE
- Existem dois modos de concorrência:
    - ResultSet.CONCUR_READ_ONLY
    - ResultSet.CONCUR_UPDATABLE
- Temos três métodos na interface Statement para executar comandos SQL:
    - boolean execute(String sql)
    - ResultSet executeQuery(String sql)
    - int executeUpdate(String sql)
- Um objeto ResultSet é usado para ler os resultados de uma consulta em formato tabular (linhas contendo as colunas especificadas). Esse objeto mantém um cursor que aponta para a linha atual, e você só pode ler uma linha por vez.
- As regras para fechar recursos JDBC são:
    - O objeto ResultSet é fechado primeiro, depois o objeto Statement, depois o Connection.
    - Um ResultSet é automaticamente fechado quando outro ResultSet é executado a partir do mesmo objeto Statement.
    - Fechar um Statement também fecha o ResultSet.
    - Fechar um Connection também fecha os objetos Statement e ResultSet.
- Para realmente obter os dados, ResultSet possui métodos getter para vários tipos de dados. Para cada método, há duas versões:
    - Uma que recebe o nome da coluna.
    - Outra que recebe o índice da coluna.
- Ao trabalhar com conjuntos de resultados roláveis (TYPE_SCROLL_INSENSITIVE ou TYPE_SCROLL_SENSITIVE), temos muitas opções para mover o cursor, como:
    - absolute(int row)
    - first()
    - previous()
    - relative(int rows)

Autoavaliação

1. Dado:

```java
public class Question_29_1 {
   public static void main(String[] args) {
      try(Connection con =
            DriverManager.getConnection(
                     "jdbc:mysql://localhost");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(
                     "SELECT * FROM user")) {
         while(rs.next()) {
            System.out.println(rs.getObject(1));
         }
      } catch(SQLException e) {
         System.out.println("SQLException");
      }
   }
}
```

Qual é o resultado se a consulta não retornar nenhum resultado?

A. SQLException
B. Nada é impresso
C. A compilação falha
D. Ocorre uma exceção não capturada em tempo de execução

---

2. Qual das seguintes é equivalente a rs.absolute(-1)?

A. rs.absolute(1);
B. rs.afterLast();
C. rs.last();
D. rs.relative(-1);

---

3. Qual das seguintes opções mostra a ordem correta para fechar os recursos do banco de dados?

A. ResultSet, Connection, Statement
B. Statement, ResultSet, Connection
C. Connection, Statement, ResultSet
D. ResultSet, Statement, Connection

---

4. Dado:

Qual é o resultado?

A. true
B. false
C. 1
D. Ocorre uma exceção em tempo de execução

---

5. Qual das seguintes pode ser uma forma válida de obter o valor da primeira coluna de uma linha?

A. rs.getInteger(1);
B. rb.getString("0");
C. rb.getObject(0);
D. rb.getBoolean(1);